CS 671 Automated Reasoning

Automatic Proof Procedures for Nuprl

1. **Decision Procedures**
2. **Rewriting**
3. **Connecting external tools**

Fully Automatic Proof Procedures

- **Solve problems in narrow application domains**
  - Translate proof goal into different problem domain
  - Use efficient algorithms for checking translated problems
  - Can be implemented in Nuprl or connected as external proof tool
- **Decision Procedures**
  - `Eq`: trivial equality reasoning (limited congruence closure algorithm)
  - `Arith`: standard, induction-free arithmetic
  - `SupInf`: solve linear inequalities over $\mathbb{Z}$
- **Rewriting: replace terms by equivalent ones**
  - Computational and definitional equality
  - Derived equivalences in lemmata and hypotheses
- **Proof Search Mechanisms**
  - JProver: intuitionistic first-order logic
  - `provePVS`: under construction

Arith: INDUCTION-FREE ARITHMETIC

- **Input sequent:** $H \vdash C_1 \vee \ldots \vee C_m$
  - C_i is an arithmetic relation over $\mathbb{Z}$
    built from $<$, $\leq$, $>$, $\geq$, $=$, $\neq$, and $\neg$
- **Theory covered:**
  - ring axioms for $+$ and $*$
  - total order axioms of $<$
  - reflexivity, symmetry and transitivity of $=$
  - limited substitutivity
- **Proof procedure:**
  - Translate sequent into a directed graph
    whose egdes are labelled with natural numbers
  - Check if the graph contains positive cycles
- **Implemented as Nuprl procedure (Lisp level)**

SupInf: LINEAR INEQUALITIES OVER $\mathbb{Z}$

- **Adaptation of Bledsoe's Sup-Inf method**
  - Complete only for the rationals
  - Sound for integers
- **Proof procedure:**
  - Convert sequent into conjunction of terms $0 \leq e_i$
    where each e_i is a linear expression over $\mathbb{Q}$ in variables $x_1 \ldots x_n$
  - Check if some assignment of values to the x_j satisfies the conjunction
  - Determine upper and lower bounds for each variable in turn
  - Identify counter-examples if no assignment exists
- **Implemented as Nuprl procedure (ML level)**

Rewriting: replace terms by equivalent ones

• Simple Rewrite Tactics

– `Fold` *name* c, `Unfold` *name* c: fold/unfold abstraction *name* in clause c

– `Subst` $t_1 = t_2 \in T$ c: substitute t_1 by t_2 in clause c

– `Reduce` c: repeatedly evaluate redices in clause c

• Nuprl's rewrite package

– Functions for creating and applying term rewrite rules

– Supports various equivalence relations

– Based on conversions and tactics for applying them to clauses in proofs

• Conversions

– Language for systematically building up rewrite rules

– Organized like tactics: atomic conversions, conversionals, advanced conversions

– Transform terms and provide justifications

– Need to be supported by various kinds of lemmata

See Section 9.9 of the Nuprl 5 manual for details

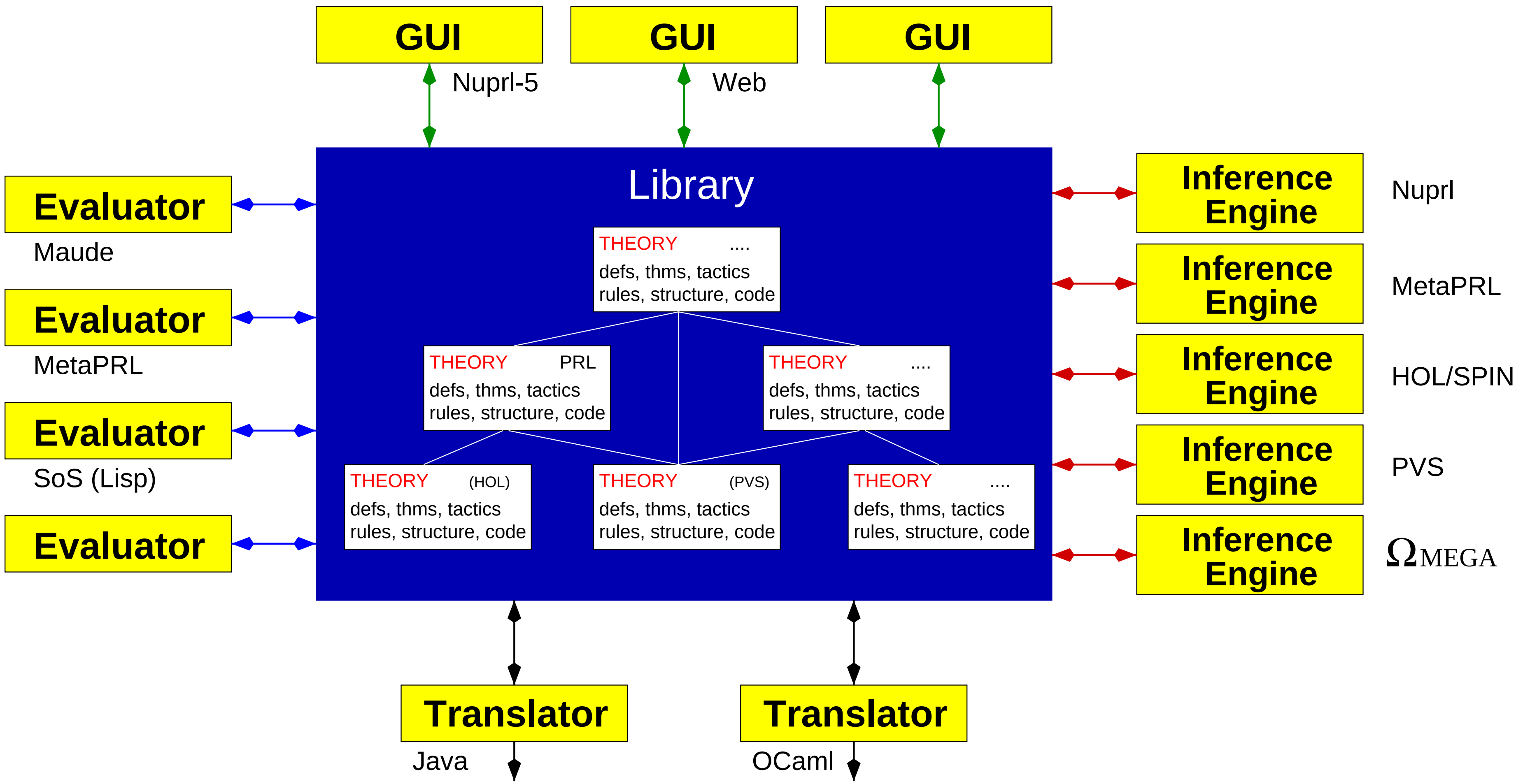
Atomic Conversions

- **Folding and Unfolding Abstractions**
  - `UnfoldC` *abs*: Unfold all occurrences of abstraction *abs*
  - `FoldC` *abs*: Fold all instances of abstraction *abs*
  - Versions for (un)folding specific instances available as well
- **Evaluating Redices**
  - `ReduceC:` contract all primitive redices
  - `AbReduceC:` contract primitive and abstract (user-defined) redices
- **Applying Lemmata and Hypotheses**
  - Universally quantified formulas with consequent $a\ r\ b$
  - `HypC` i / `LemmaC` *name*: rewrite instances of a into instances of b
  - `RevHypC` i / `RevLemmaC` *name*: rewrite instances of b into instances of a

Building Rewrite Tactics

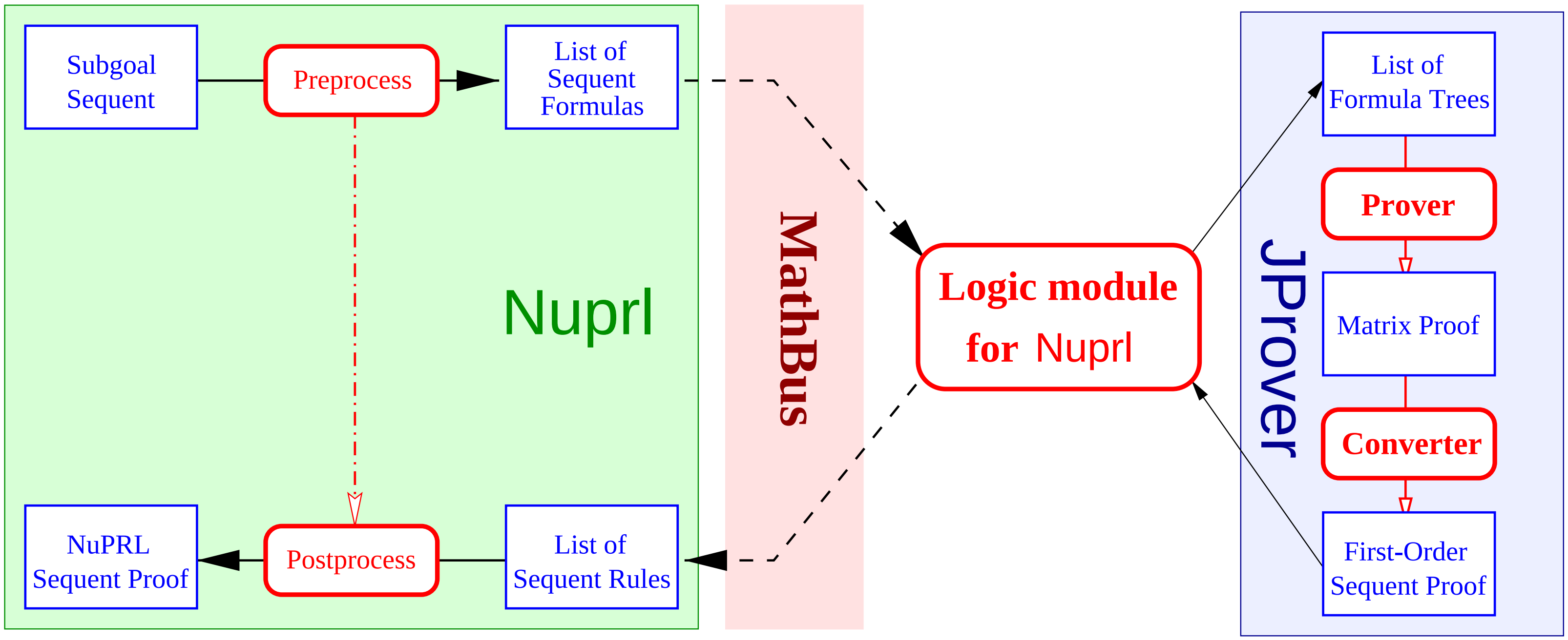
- **Construct advanced Conversions using Conversionals**
  - `ANDTHENC`, `ORTHENC`, `ORELSEC`, `RepeatC`, `ProgressC`, `TryC`
  - `SubC`, `NthSubC`, `AddrC`, `SweepUpC`, `SweepDnC`, `DepthC`
  - `AllC`, `SomeC`, `FirstC`
- **Define Macro Conversions**
  - `MacroC` *name* c_1 t_1 c_2 t_2 : Rewrite instance of t_1 into instance of t_2
    c_1 and c_2 must rewrite t_1 and t_2 into the same term (*name*: failure token)
  - `SimpleMacroC` *name* t_1 t_2 *abs* : Rewrite t_1 into t_2
    by unfolding abstractions from *abs* and contracting primitive redices
- **Transform Conversions into Tactics**
  - `Rewrite` c i: Apply conversion c to clause i
  - `RewriteType` c i: Apply c to the type of a term in clause i
  - `RWAddr` *addr* c i: Apply c to the addressed subterm of clause i
  - `RWU` / `RWD`: Apply conversion to all subterms of a clause

Digital libraries of formal algorithmic knowledge



Library as platform for cooperating reasoning tools

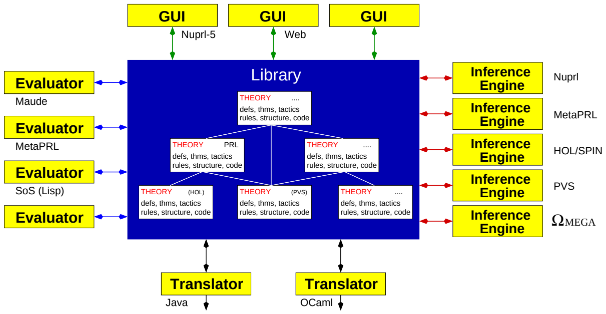
Integrating JProver and Nuprl



- JProver: Matrix prover for first-order intuitionistic logic
  + Proof Transformation: matrix proof $\longrightarrow$ sequent proof
  Stand-alone implementation in OCaml

- **Cooperation Methodology:**
  – Communication of formulas in uniform MathBus format
  – Logic module converts between Nuprl and JProver representations
  – Pre- and postprocessing in Nuprl widens range of applicability

Towards Formal Digital Libraries . . .



- **Connect**
  - Additional proof engines: PVS, HOL, MinLog, . . .
  - Multiple browsers (ASCII, web, . . .) and editors (structured, Emacs-mode, . . .)

- **Provide new features**
  - Archival capacities (documentation & certification, version control)
  - Embedding external library contents
  - A variety of justifications (levels of trust)
  - Creation of formal and textual documents
  - Asynchronous and distributed mode of operation
  - Meta-reasoning (e.g. about relations between theories)

$\Downarrow$

Authoritative reference for reliable software construction